# The CMOS 6502

*A new version of the 6502 microprocessor does more than save power—it includes powerful new instructions*

**by Steven Hendrix**

Rockwell has introduced a CMOS (complementary metal-oxide semiconductor) version of the 6502 microprocessor that fills a number of gaps in the standard 6502's instruction set while offering the low power-consumption advantages of CMOS technology. Pin and software compatible with the standard 6502 chip, the CMOS version (designated the R65C02) promises to extend the range of applications that 6502-based packages can serve.

A mainstay of the personal-computer industry since the first Apple computer was produced, the standard 6502 microprocessor has a simple, straightforward instruction set and simple interfacing requirements. The instruction set at first appears to be restricted in comparison to other 8-bit processors such as the Z80, but, in practice, the simplicity of the instruction set often yields a shorter, faster program for common microprocessor applications. The instruction set does have restrictions on the use of certain addressing modes with some instructions and has several minor anomalies that are poorly documented.

In this article I will discuss some of the 6502's lesser-known deficiencies and the changes in the CMOS version that correct some of these problems. I will also review the CMOS version's instructions and added addressing modes, and finally I will describe some hardware interfacing considerations.

**Several 6502 instructions don't behave as you might expect them to.**

## Quirks of the 6502

Several instructions on the 6502 do not behave as the documentation would have you believe. These irregularities rarely affect programs, which makes them more difficult to debug when they do enter into a program. The quirks discussed here pertain to the return-from-interrupt instruction, the branch-instruction timing, the absolute indirect-addressing mode, and bus cycles on certain index-addressing modes. The CMOS version's design has not altered the return-from-interrupt and branch-instruction timing; therefore, the information presented on these topics pertains to both the standard and CMOS versions of the 6502. The CMOS version's design, however, has corrected the absolute indirect-addressing mode and bus-cycle anomalies.

### RTI versus RTS

The RTI (return-from-interrupt) instruction appears functionally equivalent to the sequence PLP (pull status register from stack), RTS (return from subroutine). An interrupt is acknowledged at the end of an instruction, at which time the processor pushes the contents of the program counter on the stack, high byte followed by low byte, and then pushes the processor-status byte on the stack before jumping through the interrupt vector to the interrupt-handling routine.

The difference between the RTI instruction and the PLP, RTS sequence lies in the sequence in which the program counter is incremented. During a JSR (jump to subroutine), the value pushed on the stack is the address of

the third byte of the JSR instruction. Thus, the program counter is reloaded during an RTS instruction and then incremented before the attempt to fetch the next instruction. An interrupt pushes the address of the first byte of the next instruction to be executed, so the RTI instruction reloads the program counter and fetches the next instruction without first incrementing the program counter. This difference becomes especially important in writing software for tracing or single-stepping functions.

### Branch-Instruction Timing

The branch-instruction timing problem lies not with the 6502, but rather with its documentation. The original data sheets specify the timing correctly, but several independent tutorials have incorrectly stated how long a branch instruction takes.

Unlike most other 6502 instructions, a branch instruction requires a variable number of clock cycles—from two to four, depending on the circumstances surrounding the branch.

During the first clock cycle (bus cycle), the processor fetches the branch op code. The second cycle fetches the second byte of the instruction, which is the offset to be used if the branch is taken.

**Several independent tutorials have confused 6502 branch-instruction timing considerations.**

If the branch condition (flag set or cleared) is not met, the fetch for the next instruction occurs during the next clock cycle. If the branch is taken, the next cycle is used to add the offset to the low-order byte of the program counter. If there is a carry or borrow from this operation (considering the offset to be a signed value), a fourth clock cycle is used to update the high-order byte of the program counter.

The net result is that a branch that is not taken requires two clock cycles. A branch to a location within the same page requires three clock cycles, and only in the case of a branch that crosses a page boundary does the instruction require the full four cycles. Typical timing loops, especially for intervals under a millisecond or so, require close attention to these details of the branch-instruction timing.

### Absolute Indirect Mode Wraparound

The absolute indirect-addressing mode works only with the JMP (jump) instruction. In normal use, it is a 3-byte instruction: the first byte contains the op code (6C)(all instructions and addresses are specified in hexadecimal); the second byte contains the low-order part of a memory address; and the third byte contains the high-order part of that address. The processor loads the byte at the referenced address into the low half of the program counter, and it loads the byte in the next higher memory location into the high half of the program counter. Thus, the instruction's effect is to jump to the location specified by the two bytes stored at the address given in the instruction.

A problem arises, however, when the jump destination is stored with the two bytes split between two memory pages (that is, if the second byte of the instruction is FF). The processor loads the referenced byte into the low half of the program counter and attempts to increment the address given in the instruction to load the high byte. However, it disregards the carry from the increment operation on the low byte of the address, with the result that the high byte of the program counter is loaded from the memory location 255 bytes prior to the referenced location.

In table 1 the JMP instructions illustrate this problem. The left-hand-column code operates correctly, loading the value A345 into the program counter. The right-hand-column code, however, does not correctly load the value A345 into the program. It does load the value 45, stored at location 02FF, into the program counter's low-order byte, but

| | |
|---|---|
| A345 JMP (0200) | A345 JMP (02FF) |
| 0200 45 | 0200 59 |
| 0201 A3 | 02FF 45 |
| | 0300 A3 |
| Result: A345 → PC | Result: 5945 → PC |

**Table 1:** *Two sets of memory contents illustrating operation of the 6502 JMP instruction. The left-hand column of code operates as expected, but due to an instruction-set anomaly, the right-hand column's code yields an unexpected result because the program counter's desired high-order byte resides in a different page of memory than does the low-order byte.*

rather than transferring to the next page of memory to obtain the high-order program-counter byte from location 0300, it incorrectly loads the value stored at location 0200 (59 in this case) into the program counter's high-order byte.

This anomaly can cause major problems when you attempt to develop general-purpose table-driven software. If the application program does not contain special code to insure that an indirect jump never references an address at the end of a page, unpredictable behavior that is difficult to trace can result. The R65C02 reportedly handles the absolute indirect-addressing mode correctly for all cases.

### Spurious Bus-Read Cycles

A rare problem with I/O (input/output) devices can occur because of the nature of the 6502 bus. Two specific factors combine to cause this problem: all I/O is memory-mapped, and there is no such thing as an inactive bus cycle. In some cases, indexed instructions can lead to inadvertent accesses to I/O devices because of these two facts.

The 6502 treats memory and I/O ports alike, viewing both as memory. As a result, a system's decoding hardware causes I/O ports to appear at specific locations that look like part of the memory-address space to the 6502. A "read" bus cycle addressing a port acts as an "input" operation, and a "write" cycle acts as an "output" operation.

The 6502 does not have separate pins for a "read" and a "write" signal, as do other processors such as the 8080 or the Z80. Instead, the R/W (read/write) signal is used to designate a "read" cycle if it is in a high state or a "write" cycle if it is in a low state. Timing is coordinated by the Phase 2 clock. If the read/write line is high when the Phase 2 clock is high, the device whose address appears on the address bus places data on the data bus. If the read/write line is low while the Phase 2 clock is high, the addressed device accepts data from the bus.

To show how indexed instructions can interfere with I/O devices, let's examine the bus cycles carried out to load the accumulator from an absolute address indexed by the X register. In standard 6502 mnemonics, this load instruction is LDA ADDR,X. This instruction takes four cycles unless the indexing crosses page boundaries, in which case it takes five. The latter is the troublemaker.

During the first cycle, the 6502 fetches the op code. The second and third cycles are used to fetch the low and high bytes of ADDR, respectively. If the indexing operation does not cross a page boundary, the sum of ADDR and X is placed on the address bus during the next cycle, and the A register is loaded from the data bus, finishing the instruction. If a page boundary is crossed, however, a partially formed address is placed on the bus during cycle four and the actual load happens in a fifth cycle. For normal memory access, the fifth cycle does no harm because it is a read cycle, resulting in memory placing data on the bus but no registers or memory being changed by it. (Even if the instruction is a store instruction, the cycle involving this partially formed address is a read cycle.)

Certain I/O devices, however, are affected by read operations. For instance, a spurious read from a 6850 ACIA (Asychronous Communications Interface Adapter) could reset the "receive data register full" flag, so that a later operation would find that data was not available. Various other I/O devices such as parallel ports and

counter/timers can also be affected by spurious reads. If the indexed address crosses a page boundary from the page in which the I/O device resides, the partially formed address placed on the bus during the fourth bus cycle can trip the I/O device. The R65C02 reportedly corrects this problem.

## New Instructions

The R65C02 includes a number of new instructions, making it more powerful than the 6502. (The text box "An Assembler for the R65C02" on page 452 describes an assembler that supports the R65C02's extended instruction set.) Conditional branching based on the state of any bit in page 0, an unconditional short relative branch, stack operations for the X and Y registers, the ability to set or clear any individual bit in page 0, zeroing any byte in memory, and a "test and reset" or "test and set" memory bit instruction have been added.

The BBRx (branch on bit reset) instructions permit any bit in page 0 to be used as a flag. These are 3-byte instructions, with the op code in the first byte, the page-0 address of the byte containing the flag in the second byte, and the relative jump displacement in the third byte. Bits 6 through 4 of the op code give (in binary) the number of the bit within the page-0 byte to be tested. The processor reads the byte from page 0, checks the bit designated by the op code, and continues normal program flow if the designated bit is a 1. If it is a 0, a normal signed relative short branch is executed, using the third byte of the instruction for the offset. The BBSx (branch on bit set) instructions do the same thing except that they take the branch only if the referenced bit is set to 1.

**The R65C02 includes a number of new instructions, making it more powerful than the standard 6502.**

## Unconditional Short Branch

The unconditional short-branch instruction (BRA) eases writing of position-independent code and in some cases allows shorter code. With the 6502, a sequence such as SEC (set carry), BCS (branch if carry set) is sometimes necessary to cause an unconditional position-independent jump. Even that sequence requires 3 bytes, as does a normal absolute jump (JMP). The BRA instruction permits an unconditional, position-independent branch requiring only 2 bytes.

Four new stack-manipulation instructions have been added to act on the X and Y registers. In 6502 programs, the X and Y registers could be pushed only by transferring them first to the A register. Thus, the normal sequence for saving the registers for an interrupt routine went something like this: PHA (push the A

register on the stack), TXA (transfer X to A), PHA, TYA (transfer Y to A), PHA. This sequence required extra time and memory and also made it difficult for a routine to save and restore all the registers and make use of a value passed to it in the A register. The four new instructions permit direct pushing and pulling of both the X and Y registers.

### Set and Clear Page-0 Bits

Companions to the BBRx and BBSx instructions, the RMBx (reset-memory bit) and SMBx (set-memory bit) instructions permit setting and clearing single-bit flags in page 0 without affecting any internal processor registers accessible to the programmer. As before, bits 6 through 4 of the op code specify which bit is affected, and the second byte of the instruction specifies the page-0 location affected.

The new STZ (store zero) instruction permits zeroing an entire byte anywhere in memory without affecting processor registers. Four available addressing modes allow a 2-byte form for page-0 operations and a 3-byte form for general addresses, either of which may be indexed by the X register.

The TRB (test and reset bits) instruction is a composite of the 6502 BIT (bit test) and AND (logical and) instructions. The N (negative) flag is set to the value of bit 7 of the referenced memory location, and the V (overflow) flag is set to the value of bit 6. A logical AND is then performed between the referenced memory location and the A register, with the result stored into the memory location (A is unaffected), and the Z (zero) flag is changed to indicate the result of this operation (set if the result is 0, reset if it is nonzero). Note that, just as on the 6502, the N and V flags pertain to the value in memory *before* the AND operation takes place. The TSB (test and set bits) instruction is similar except that a logical OR is substituted for the logical AND operation.

### Addressing Modes

In addition to totally new instructions, the R65C02 enables some existing addressing modes to be used with instructions that did not accept those modes on the original 6502. It also adds an entirely new addressing mode usable with a number of present instructions that should prove useful in making better use of the processor registers.

The 6502 has no simple indirect-addressing mode other than the JMP instruction. With no 16-bit registers to hold addresses, 6502 programs frequently keep addresses in page 0, especially when passing addresses to and from subroutines. However, the only way to use those addresses to access the data to which they point is through the pre- or post-indexed indirect-addressing modes. Thus, a common sequence in programs consists of loading the Y register with 0, followed by an operation using the "indirect, indexed by Y" addressing mode. Not only does this sequence result in extra code requiring additional memory space and execution time, but it ties up the Y register, which might be better used in other ways.

**The R65C02 includes a simple indirect-addressing mode using a 2-byte address.**

The R65C02 corrects this deficiency by adding a simple indirect-addressing mode, which uses a 2-byte address stored in page 0. This addressing mode can be used with all the major accumulator instructions: ADC (add with carry), AND (logical and), CMP (compare memory with accumulator), EOR (logical exclusive-or), LDA (load accumulator from memory), ORA (logical inclusive-or), SBC (subtract with borrow), and STA (store accumulator to memory).

### New Modes for BIT

The BIT (bit test) instruction of the 6502 is severely limited in addressing modes. This instruction accepts only two modes: absolute (direct) and 0 page. Because this instruction functions as a logical AND except that the result is discarded, it is normally used to test flags. Most such tests would be most conveniently done with an immediate addressing mode, which is not permitted. Instead, 6502 programs must use a backward form of logic, loading the test mask using the immediate mode and then doing the test on the data directly from memory.

The R65C02 BIT instruction permits additional addressing modes—immediate, 0-page indexed, and absolute indexed. These added modes cover the vast majority of the situations in which this instruction would be used.

### Increment and Decrement Accumulator

Arithmetic on the X and Y registers is not permitted by the 6502; neither is incrementing or decrementing the accumulator. Though such a need is rare, it does arise, and the lack of an accumulator-addressing mode for the increment and decrement instruction results in various kludges to get the desired result. Three alternate ways are commonly used. The most obvious is to use the ADC (add with carry) instruction to add an immediate value of 1. Because the 6502 does not provide a simple "add" instruction (without carry), this alternate method also requires a preceding CLC (clear carry) instruction, unless the state of the carry bit from prior operations is known. Alternatively, setting the carry bit followed by adding an immediate value of 0 accomplishes the same thing.

If the X or Y registers are not in use at the particular point in the program, it is possible to transfer the value from the A register to one of those registers and take advantage of the increment or decrement instructions for X and Y. A third method, most commonly used when the next step is to store the accumulator value in memory, is to store the A register value first and then increment it in memory, because the INC (increment) and DEC (decrement) instructions accept several different addressing modes for operations directly on data in memory.

The R65C02 eliminates all of this foolishness by allowing the accumulator-addressing mode to be used with the increment and decrement instructions, enabling them to operate on all three of the general-purpose registers.

### Hardware Factors

The R65C02 has the electrical characteristics you would expect from the current generation of CMOS integrated circuits. Versions for speeds to 6 MHz will probably be available. Power consumption is low and varies with speed, as is normal for CMOS technology. With the clock stopped, 10 $\mu$W power consumption is listed as maximum. Maximum power consumption in normal operation is listed as 4 mA (20 mW) per MHz, making battery-powered operation feasible when this chip is combined with the new CMOS memory chips.

Rockwell claims that the basic R65C02 version is pin and software compatible with the 6502. Another version, the R65C102, can generate all clock signals on-chip; it needs only an external TTL (transistor-transistor logic) level single-phase clock input (as does the 6502) or an external RC network or crystal. It also has a quadrature clock output, which is not provided by the 6502. This clock goes high in the middle of the phase-1 clock and returns low in the middle of the phase-2 clock.

The 6502 has not been commonly used in applications requiring multiple processors or direct-memory access, largely because it cannot float its address bus. Both the R65C102 and another version, the R65C112, have signals to permit bus sharing. The bus-enable (BE) signal permits an external device to cause the processor to float the address and data buses and the R/W signal, permitting access to the system buses. To prevent bus arbitration from interfering with read-modify-write instructions such as shifts and increments, a memory-lock (ML) output signal is provided to notify external devices that the processor cannot relinquish the bus until completion of the instruction. The R65C112 is designed to be used as a slave processor, requiring a two-phase clock input that would be generated by the system master processor.

WDC's W65C02S HAS ML & BE

### Summary

The CMOS version of the 6502 chip fills in a number of gaps in the 6502 instruction set in addition to adding the obvious advantages of CMOS technology. The characteristics of the new chip permit the 6502 to expand in both directions into areas that were previously impractical. Completely battery-powered systems are now feasible for small, dedicated applications. Additionally, the added bus control permits multiple-processor systems and sophisticated direct-memory-access schemes to be used with this processor.

Perhaps the most impressive feature of the CMOS version is complete compatibility with the 6502 specifications, permitting the enormous base of 6502-based hardware and software to be used with the newer processor. The R65C02 processor represents a step above the 6502 similar to the step from the 6800 to the 6502, without the accompanying compatibility problems. The current popularity of 6502-based personal computers provides a large market for new applications of this processor. ■

**An Assembler for the R65C02**

*HEXASM is a full-feature resident assembler that supports the R65C02 microprocessor's extended instruction set. In addition to including such features as macros, conditional-assembly, and source-file-chaining functions, it can optionally be configured to either accept or reject constructs that are unique to the R65C02. HEXASM runs under HEXDOS on Ohio Scientific's OSI C1P and is available for $38.50 from Hx Computer Products, Route 8, Box 81E, New Braunfels, TX 78130 or The 6502 Program Exchange, 2920 West Moana, Reno, NV 89509.*

*Steve Hendrix, an instructor pilot for the U.S. Air Force, has a B.S. in computer science and mathematics from the USAF Academy. He can be reached at Route 8, Box 81E, New Braunfels, TX 78130. His hobbies include computers and astronomy.*